Data Visualization via the Integration of Java Technologies

Vincent H. Pell, David G. Fout and Kenneth B. Sall
Century Computing Division of AppNet, Inc.
8101 Sandy Spring Rd.
Laurel, MD 20707, USA

Matt Brandt
NASA/Goddard Space Flight Center
Mailstop 588.0
Greenbelt, MD 20771, USA

vpell@CenturyComputing.com, dfout@CenturyComputing.com, ksall@CenturyComputing.com, Matthew.Brandt@gsfc.nasa.gov

Abstract

NASA/GSFC and AppNet, Inc. have developed a web-based information visualization toolkit that allows developers to quickly and easily add advanced visualization capabilities to new, as well as to existing applications. The goal of the Visual Analysis Graphical Environment (VisAGE) is to provide a set of data-centric components that work together in a robust framework that supports the addition of sophisticated visualization capabilities to web-based and client-server applications. VisAGE provides both server-side and client-side APIs. The server is often used with time-based, streaming data sources. The client-side interfaces can be used without a VisAGE server, such as for generic visualization components.

This paper discusses a variety of Java and web technologies that made VisAGE possible. Pitfalls and insights are highlighted.

Keywords: Java2D, Java3D, RMI, XML, Data Visualization

Data Visualization via the Integration of Java Technologies

Background: VET and VisAGE

The NASA/GSFC project called the **Virtual Environment Testbed (VET)**, promotes the exploration of virtual environment technologies within the spacecraft mission operations domain. The goals of this effort are to investigate virtual environments as they apply to the mission operations domain and to transfer these technologies into operations.

In order to reduce the growing trend of increasing software development costs and schedules, NASA/Goddard Space Flight Center and the Century Computing Division of AppNet, Inc. have developed a web-based information visualization toolkit called **Visual Analysis Graphical Environment (VisAGE)** that allows developers to quickly and easily add advanced visualization capabilities to new, as well as to existing applications. Current visualizations include: 2D and 3D bar charts, 2D and 3D strip charts, 2D pie charts, 2D histograms, 3D carousel, text and satellite visualizations [large screenshot]. The goal of VisAGE is to provide a set of data-centric components that work together in a robust framework that supports the addition of sophisticated visualization capabilities to web-based and client-server applications.

VisAGE provides a front end for time-based, streaming data sources (e.g., telemetry, stock information, etc.). The toolkit is used to provide different views of the data. VisAGE employs a three tiered architecture. The middle tier, the VisAGE server, is essentially a framework that can be adapted to a specific data source. With this design, inter-process communication between VisAGE clients and servers is an important consideration. VisAGE provides both server-side and client-side APIs. A VisAGE server is a Java application that provides remote data services for VisAGE clients. The APIs allow developers to easily create distributed platform independent data visualization applications. The API consists of two parts, a client-side API for building VisAGE clients, and a server-side API for building VisAGE servers. Remote Method Invocation (RMI) is used as the communication mechanism between VisAGE clients and VisAGE servers.

The APIs make it possible to determine what data exists (both sources and individual data points) and then to issue requests for blocks of data. Visualizations are associated with particular data sources; when a client receives the data, it can display it using different visualizations. It is also possible to define a database as a data source, even though it is not time-based. The client-side interfaces can also be used without a VisAGE server, such as for generic visualization components. Visualization components use the Model/View/Controller paradigm, so VisAGE enables users to dynamically define different views of data in real-time.

This paper discusses a variety of Java and web technologies that were investigated in the development of VisAGE, namely installation, security, Java2D, Java3D, RMI, Swing, JavaBeans, and XML. Pitfalls and insights are highlighted.

Installation and Java Security Issues

Delivering a complex Java application to an end user poses several problems. Our installation includes the VisAGE application, the Java 2 Platform's Java Runtime Environment (JRE) and Java Plug-in (which allows Java 2 to be executed from a web browser), Java3D and the JavaHelp API. By simply supplying a JAR file for the user to install and run, we would be assuming that the user understands how to launch the VM, knows how to specify a classpath, and knows which Java extensions need to be installed beforehand. However, from a user perspective, what is really desired is to obtain a self-extracting executable that when launched will completely install all components necessary to run the application, and perhaps will create a shortcut somewhere on the desktop that will launch the application via a mouse click. Ideally, the user should be able to visit a web site where he can download the full installer, or simply get another installer that patches or updates an older version of the application. Furthermore, Java security details are too complex for many users; therefore, we provided a custom Security Configuration Tool. The details of our investigation of installation and security issues follow.

Applets vs. Application

One of the earliest issues the development team struggled with was the tradeoffs between Java applets and applications. Java applets (which run in a web browser) are loaded over a network and are restricted to the Java "sandbox". Applets are prevented from reading and writing files on the client file system, cannot make network connections except to the originating host, and are not allowed to accept connections from any host. They are prevented from starting other programs on the client, are not allowed to load libraries, or to define native method calls. In general, applets have no access to the client host. However, the restricted functionality of applets can be eliminated if the applet is digitally signed. Both Netscape and Internet Explorer recognize digital certificates.

Digital signatures can eliminate the problem of restricted access that limit applets. However, there are issues with using applets that need to be considered when choosing between Java applets or Java applications. Applets must be downloaded each time they are used. The larger the applet, the longer it takes to start. Downloading the applet does take time, but it also insures that the client is always using the latest version of the software. Another potential problem is that web browsers often lag behind in their support of the latest Java version. JavaSoft has addressed this problem by creating the Java Plug-in, a Web browser plug-in that supports new Java versions as soon as they are available.

Using Java applications presented many of the same issues developers encounter when using applications written in other languages. Java applications must be installed on the user's computer. However, once installed, the application is always instantaneously available to the client. Keeping clients up-to-date with the latest version of the software can also be problematic. When the application is updated by developers, the end user must be notified that a new version is available, obtain it, and re-install the software. Therefore, as VisAGE is launched, it automatically checks whether an update is available from the web site and, if so, optionally downloads the update. For all of the above reasons, the VisAGE team decided to provide both applet and application installation options.

InstallAnywhere vs. InstallShield

As noted above, ease of deployment and installation was a primary concern. We first chose InstallAnywhere (ZeroG Software) which generated HTML and CGI scripts to launch or download the installer from the web for all major platforms. The user simply visits a web site created by InstallAnywhere and clicks "install" to place the application on his system without needing to be concerned about prerequisite software such as the Java VM. InstallAnywhere provides extensive controls for the developer to specify exactly what to deliver, where to place it on the user's system, which versions are needed for different platform, etc. It provides a simple GUI interface that can either quickly build an installer by using the "wizard" or can expose complex features when used in advanced mode. Although InstallAnywhere worked wonderfully when VisAGE was implemented in JDK1.1.x, it failed to cope with JDK1.2 and Java Extensions. The newer version, InstallAnywhere 2.5, does support JDK1.2, but cannot deliver 1.2 VM. Also, there was no way to detect and install Java Extensions that were needed by VisAGE, such as Java3D.

We also considered InstallShield Java edition, but ultimately decided against it. Although InstallShield was very similar to InstallAnywhere in capabilities, it assumed that the user already had a Java VM installed. We reluctantly moved to InstallShield Professional which gave us tremendous capabilities to inspect installed components and tailor our installer to deliver exactly what our users needed. InstallShield Pro gave us the ability to launch other installers within the installation of VisAGE. When the installer for components like a Java VM or Java extension is run, it registers the components in the Windows system registry. On the other hand, when those components are installed by simply placing necessary files on the hard disk (the InstallAnywhere approach), the registration process is bypassed. The registry approach enabled us to install JRE1.2, Java3D, and even to set up the Java Plug-in to run JDK1.2 applets on the user's system. However, InstallShield Professional is Windows-specific, a limitation that is often not acceptable for Java applications. The other drawback was the inherent complexity of the tool needed to build the installer since is was not specifically designed for packaging Java applications. Also, it required a developer to learn InstallShield's own scripting language to implement many desired features. We ultimately deployed a Windows-specific installer and included online instructions for Unix installation and configuration.

Security Configuration Tool

Since the project's inception, there has been considerable interest in having VisAGE run as both an application and an applet. Unfortunately, VisAGE requires functionality that is restricted or prohibited by the Java virtual machine when running in an applet environment. The team investigated three separate but incompatible implementations of the Java security model: the Netscape Communicator implementation, the Microsoft Internet Explorer implementation, and the Sun implementation. The Netscape and Microsoft implementations were based on the earlier JDK 1.1 security framework, whereas the Sun implementation we investigated was a complete security model included in JDK 1.2. Since Sun's solution works with any browser if used in conjunction with the Java Plug-in, it became our choice for a security model. However, the Sun implementation requires the user to know about the applet's security certificate as well as what privileges the applet needs *before* the applet can be executed. The VisAGE developers felt that the tools provided by Sun to setup security were too complicated for our target user base. The solution was to provide a custom security configuration tool that is downloaded as part of the VisAGE download. The custom tool automatically establishes the security access needed by the VisAGE applet.

RMI for Inter-Process Communication

Java's Remote Method Invocation (RMI) was chosen as the mechanism for inter-process communication between clients and servers. Two other solutions considered were to use sockets directly with a custom protocol, or to use an implementation of the Common Object Request Broker (CORBA).

The Java language has built-in support for sockets. The chief advantage to using sockets, rather than a higher level mechanism like RMI or CORBA, is performance. However, if sockets were used a custom protocol would need to be produced and a parser for that protocol would have to be written. Therefore, a socket only implementation would probably be expensive to develop. CORBA is the Object Management Group's answer to inter-process communication. It is much more flexible than RMI because it allows for interoperability among applications written in different languages. The two biggest drawbacks in using CORBA are complexity and cost; the latter was the chief reason for not choosing CORBA for our project at this time. We also note that some ORB implementations do not perform exactly as advertised.

RMI, unlike CORBA, is very easy to use since all objects are Java objects; remote method invocations look just like local method invocations. As of JDK1.1, RMI is part of core Java and is therefore free. RMI is ordinarily only considered when the server and client are both written in Java. However, a new standard extension to Java, RMI over IIOP (RMI/IIOP), makes RMI Interfaces CORBA compatible. Finally, RMI performance is similar to that of CORBA.

Java2D vs. AWT

Initially in VisAGE 1.0, 2D visualizations such as bar charts, strip charts, and pie charts were implemented using a COTS product call JChart, developed by the KLGroup. We found that JChart enabled us to quickly generate displays for our visualizations because it supplied a framework of customizable GUI JavaBean components designed specifically for viewing numeric data arrays. However, we found that the functionality was not extensible. Each chart had a fixed, uniform behavior; it was not possible to augment the existing functionality with new behavior.

In Visage2.0, nearly half the visualizations were implemented by using Java2D to develop custom charts. Java2D greatly enhanced the graphics capabilities of standard Java since it implements many advanced graphical features that are not found in AWT (Java's basic Abstract Window Toolkit). However, Java2D does not preclude the use of AWT; it simply gives developers control in adjust rendering qualities such as interpolation, anti-aliasing, composition, and affine transformations. It also provides an extensive framework including geometry management, image processing, text rendering, and printing.

One disadvantage of using Java2D is its availability. It can only be used on the Java 2 Platform and cannot be integrated with earlier versions of the Java VM. At this time, this restricts VisAGE to Windows 95/98/NT and Solaris. Another minor problem with Java2D is its performance versus the performance of AWT. The performance degradation is mainly due to a slower drawImage method implemented by Java2D. The degradation is not noticeable by the VisAGE user at all; it is only significant when displaying very large images. (It is worth noting that Java2D at best does not perform any faster than AWT when it comes to very low level rendering.) The greatest benefit of Java2D derives from its use of a floating point coordinate system. Pixel rounding errors in VisAGE 1.0 were eliminated because Java2D handled the interpolation.

3D Content: Java3D vs. VRML2/EAI

VisAGE 1.0 used the Virtual Reality Modeling Language version 2 (VRML2) to create dynamic 3D visualizations. At that time, VRML2 was the most viable solution for adding 3D content to a Java applet. During the requirements phase of VisAGE 2.0, however, an early access version of Java3D became available. Upon evaluating Java3D it was decided that it was a better solution for 3D graphical applets and applications.

With VRML2, the applet communicated with the CosmoPlayer browser plug-in via the External Authoring Interface (EAI). This interface defines a set of methods that an external entity can perform on the VRML browser to affect the VRML scene. For simple interactions, this mechanism is very useful. However, for more complex operations, EAI proved to be cumbersome and added an unnecessary level of abstraction. Java3D, on the other hand, is written in Java, enabling 3D content to be written directly in Java code. Therefore, Java3D removes the unreliability of communicating through EAI. VRML2 was designed for content creators and thus simplicity was a key goal, at the expense of limiting functionality. For example, in VRML2 fine grain control over all aspects of the 3D scene is not possible. Java3D was created with the software developer in mind. The API provides a rich set of features for creating and manipulating a 3D scene. This makes Java3D much more preferable than VRML/EAI if complex interaction is needed.

VRML plug-ins like CosmoPlayer must run within a web browser. However, one of the objectives of VisAGE was to create 3D visualizations components that can be used like any other Java component, specifically within a Java Panel or Frame. This goal could not be met using VRML2/EAI but could with Java3D. In fact, all of the VisAGE 2.0 visualizations are now reusable JavaBeans.

Java3D is a standard extension to Java that gives developers the ability to write applets and applications that use 3D graphics. Most of our development team had prior experience with the Silicon Graphics OpenInventor toolkit. Thus, the transition to Java3D was fairly easy. Java3D eliminated the shortcomings of the VRML/EAI solution.

Swing vs. AWT

Swing is Java's new lightweight GUI toolkit; it is not based on OS-specific rendering of GUI components, as is its predecessor, AWT. Therefore, Swing is not limited to providing a "least common denominator" set of components across platforms (as is AWT). A more sophisticated set of components is supported by Swing. Despite its bugs, Swing's expanded GUI toolkit provided VisAGE with a better look and feel than what was possible using AWT. In particular, the addition of the Swing Tree, Table, and Tabbed Pane added significantly to the usability of the UI.

Unfortunately, the flexibility and extensibility of Swing are at the expense of speed. AWT components out-perform Swing components because in the case of AWT, the OS handles the rendering directly. Swing can't rely on native methods, so it must handle the widgets programmatically in Java. Although 75% of the GUI was created using Swing widgets, the overall performance of the GUI was judged acceptable.

A major problem was in the interaction of the lightweight Swing components with the heavyweight AWT components; Swing components cannot cover AWT components. For example, when Swing pulldown menus where dropped down over an AWT panel, the menu was clipped by the panel, making it impossible to select menu choices. The solution was to call

```
JPopupMenu.setDefaultLightWeightPopupEnabled(false);
```

prior to menu creation.

JavaBeans - Reusable Software Components

JavaBeans is the component architecture used by Java. JavaSoft defines a Java Bean as a reusable software component that can be manipulated visually in a builder tool, but they are actually much more than that. JavaBeans are characterized by introspection, customization, events, properties, and persistence. Introspection is the runtime analysis of what a Bean can do. Persistence means that the state of a Bean can be saved and reconstituted later. Customization means that the Bean's properties can be modified at runtime. Properties are attributes that enable developers to customize beans to a large degree.

All of the visualizations in VisAGE are JavaBeans (although JavaBeans need not be visual). This implementation decision was extremely advantageous. Our components have been remarkably reusable and have made integration with other systems easier than expected.

XML for Data Sources

Previous versions of VisAGE required that descriptions of the data sources were hardcoded in the server. Whenever a data source changed, server code had to be modified and VisAGE needed to be recompiled. This inefficiency was addressed in the latest version by using an XML file to describe the data sources. Use of an XML Source Description file reduces (or eliminates) the need for an end user to write any code when properties of one of their data sources change or when a new source is added. Instead, editing the Source Description file is all that is required. It is a plus that the XML format looks very familiar to anyone who has written HTML.

We chose to use Sun's XML library called Java Project X since it was a pure Java solution that provided a fast, robust set of features for only a modest increase in VisAGE's size. Java Project X provides core XML capabilities including a fast XML parser with optional validation and an in-memory object model tree that supports the W3C Document Object Model (DOM) Level 1 recommendation.

Summary and Conclusions

This paper discussed a number of Java technologies that were effectively used to develop state-of-the-art data visualizations. Other Java technologies including JavaHelp, Sound, and tools such as JBuilder were used, but space does not permit their discussion.

NASA is currently considering using VisAGE to incorporate advanced visualization capabilities into ongoing and future missions. Several prototype efforts are underway, such as developing a web-based front-end to the Microwave Anistropy Probe (MAP) ground system. (MAP is a Mid-Class Explorer (MIDEX) class mission designed to probe conditions in the early universe by measuring temperature differences in the cosmic microwave background radiation.)

The Java platform (especially Java 2) provided the VisAGE team all the tools needed to develop a powerful, platform-independent, web-based visualization system. While working with cutting edge technology certainly presents significant challenges, in the end it was worth the effort. Although some of the advanced Java APIs and standard extensions are relatively new at the time of this writing, they were stable enough, on the whole, for us to complete the VisAGE 2.0 release. The development team is looking forward to applying emerging technologies such as the Java Advanced Imaging, Media Framework, Sound, Speech, Shared Data and Telephony APIs in future efforts.

References

Project Links

- Virtual Environment Testbed (VET) Home Page
- VisAGE Home Page
- VisAGE Screenshot (large)
- Information Visualization
- Microwave Anistropy Probe (MAP)

Technology Links

- JavaSoft Products and APIs
- JDK 1.2 (Java2 platform)
- Explore Java2 features (overview)
- Java Plug-in (part of Java2 platform)
- Java Security
- JavaBeans
- RMI
- Java2D (2D demo)
- Java3D
- Java Media APIs (Advanced Imaging, Media Framework, Sound, Speech, Shared Data, Telephony, etc.)
- Swing
- JavaHelp
- XML: Sun's Java Project X (requires free login)
- InstallShield (InstallShield Software Corporation)
- InstallAnywhere (ZeroG Software)
- JClass (KLGroup)